A Structure Preserving Database Encryption Scheme

Yuval Elovici[1], Ronen Waisenberg[1], Erez Shmueli[1], Ehud Gudes[2]

[1] Ben-Gurion University of the Negev, Faculty of Engineering, Department of Information Systems Engineering, Postfach 653,
84105 Beer-Sheva, Israel
{elovici, ronenwai, erezshmu}@bgu.ac.il

[2] Ben-Gurion University of the Negev, Department of Computer Science, Postfach 653,
84105 Beer-Sheva, Israel
ehud@cs.bgu.ac.il

Abstract. A new simple and efficient database encryption scheme is presented. The new scheme enables encrypting the entire content of the database without changing its structure. In addition, the scheme suggests how to convert the conventional database index to a secure index on the encrypted database so that the time complexity of all queries is maintained. No one with access to the encrypted database can learn anything about its content without having the encryption key.

1 Introduction

Database is an integral part of almost every information system. According to [1] the key features that databases propose are shared access, minimal redundancy, data consistency, data integrity and controlled access.

The case where databases hold critical and sensitive information is not rare, therefore an adequate level of protection to database content has to be provided. Database security methods can be divided into four layers [2]: physical security [3], operating system security [4, 5, 6], DBMS security [7, 8, 9] and data encryption [10, 11, 12]. The first three layers alone are not sufficient to guarantee the security of the database since the database data is kept in a readable form [13]. Anyone having access to the database including the DBA (Database Administrator), is capable of reading the data. In addition, the data is backed up frequently so access to the backed up data also needs to be controlled [14]. Moreover, a distributed database system makes it harder to control the disclosure of the data.

Database encryption introduces an additional security layer to the first three layers mentioned above. It conceals the readable form of sensitive information even if the database is compromised. Thus, anyone who manages to bypass the conventional database security layers (e.g., an intruder) or a DBA, is unable to read the sensitive information without the encryption key. Furthermore, encryption can be used to maintain data integrity so that any unauthorized changes of the data can easily be detected.

Database encryption can be implemented at different levels [14]: tables, columns, rows and cells. Encrypting the whole table, column or row entails the decryption of the whole table, column or row respectively when a query is executed. Therefore, an implementation which decrypts only the data of interest is preferred.

The database encryption scheme presented in [13] is based on the Chinese-Reminder theorem where each row is encrypted using different sub-keys for different cells. This scheme enables encryption at the level of rows and decryption at the level of cells. The database encryption scheme presented in [14] extends the encryption scheme presented in [13] by supporting multilayer access control. It classifies subjects and objects into distinct security classes. The security classes are ordered in a hierarchy such that an object with a particular security class can be accessed only by subjects in the same or a higher security class. In this scheme, each row is encrypted with sub-keys according to the security class of its cells. One disadvantage of both schemes is that the basic element in the database is a row and not a cell, thus the structure of the database needs to be changed. In addition, both schemes require re-encrypting the whole row when a cell value is modified.

The conventional way to provide an efficient execution of database queries is by using indexes, but indexes in an encrypted database raise the question of how to construct the index so that no information about the database content is revealed [15, 16].

The indexing scheme provided in [17] is based on encrypting the whole row and assigning a set identifier to each value in this row. When searching a specific value its set identifier is calculated and then passed to the server which in turn returns to the client a collection of all rows with values assigned to the same set. Finally, the client searches the specific value in the returned collection and retrieves the desired rows. However, in this scheme, equal values are always assigned to the same set, thus some information is revealed when applying statistical attacks.

The indexing scheme provided in [18] is based on constructing the index on the plaintext values and encrypting each page of the index separately. Whenever a specific page of the index is needed for processing a query, it is loaded into memory and decrypted. Since the uniform encryption of all pages is likely to provide many cipher breaking clues, the indexing scheme provided in [19] suggests encrypting each index page using a different key depending on the page number. However, these schemes being implemented at the level of the operating system are not satisfactory.

Assuming the index is implemented as a B+-Tree, encrypting each of its fields separately would reveal the ordering relationship between the ciphertext values. The indexing scheme provided in [15] suggests encrypting each node of the B+-Tree as a whole. However, since references between the B+-Tree nodes are encrypted together with the index values, the index structure is concealed.

In order to overcome the shortcomings of existing database encryption schemes, a new simple and efficient scheme for database encryption is proposed which suggests how to encrypt the entire content of the database without changing its structure. This property allows the DBA to continue managing the database without being able to view or manipulate the database content. Moreover, anyone gaining access to the database can learn nothing about its content without the encryption key. The new scheme suggests how to construct a secure index on the encrypted database so that the

time complexity of all queries is maintained. Since the database structure remains the same no changes are imposed on the queries.

The remainder of the paper is structured as follows: in section 2 the desired properties of a database encryption scheme are outlined; in section 3 the new database encryption scheme is illustrated; in section 4 the desired properties of a secure indexing scheme are described; in section 5 a new indexing scheme for the encrypted database is proposed; in section 6 performance and implementation issues are discussed, and section 7 presents our conclusions.

2 The Desired Properties of a Database Encryption Scheme

According to [13], a database encryption scheme should meet the following requirements:

1) The encryption scheme should either be theoretically or computationally secure (require a high work factor to break it).
2) Encryption and decryption should be fast enough so as not to degrade system performance.
3) The encrypted data should not have a significantly greater volume than the unencrypted data.
4) Decryption of a record should not depend on other records.
5) Encrypting different columns under different keys should be possible.
6) The encryption scheme should protect against patterns matching and substitution of encrypted values attacks.
7) Modifying data by an unauthorized user should be noticed at decryption time.
8) Recovering information from partial records (records where some cells have null values) should be the same as from full records.
9) The security mechanism should be flexible and not entail any change in the structure of the database.

A naïve approach for database encryption is to encrypt each cell separately but this approach has several drawbacks. First, two equal plaintext values are encrypted to equal ciphertext values.

$$V_1 = V_2 \longleftrightarrow E_k(V_1) = E_k(V_2) \quad \textbf{(1)}$$

Therefore, it is possible, for example, to collect statistical information as to how many different values a specified column currently has, and what are their frequencies. The same holds for the ability to execute a join operation between two tables and collect information from the results. Second, it is possible to switch unnoticed between two ciphertext values. Different ciphertext values for equal plaintext values can be achieved using a polyalphabetic cipher (e.g. Vernam). However, in this solution decryption of a record depends on other records and thus requirement 4 is violated.

In the next section a new database encryption scheme complying with all the above requirements is presented.

3 A New Database Encryption Scheme

The position of a cell in the database is unique and can be identified using the triplet that includes its Table ID, Row ID, and Column ID. We will refer to this triplet as the cell coordinates.

We suggest a new database encryption scheme where each database value is encrypted with its unique cell coordinates. These coordinates are used in order to break the correlation between ciphertext and plaintext values in an encrypted database. The new scheme has two immediate advantages. First, it eliminates substitution attacks attempting to switch encrypted values. Second, patterns matching attacks attempting to gather statistics based on the database encrypted values would fail.

a) Table T before Encryption

Row	C
0	16
1	85
2	37
3	16
4	16
5	92
6	37
7	50
8	24
9	86

b) Encryption of Table T Using the Naive Approach

Row	C
0	#$
1	]{
2	&*
3	#$
4	#$
5	^%
6	&*
7	0-
8	+=
9	@!

c) Encryption of Table T Using the New Scheme

Row	C
0	!#
1	:]
2	&*
3	"/
4	~?
5	|^
6	>\
7	@=
8	){
9	-+

Fig. 1. Database encryption using two approaches.

Figure 1 illustrates database encryption using two approaches. Figure 1a describes a database table (T) with one data column (C). Figure 1b describes encryption of table T using the naïve approach. Figure 1c describes encryption of table T using the new approach where each cell is encrypted with its cell coordinates. It is easy to see that equal plaintext values in figure 1a are encrypted to different ciphertext values in figure 1c as opposed to the ciphertext values in figure 1b.

3.1 Encryption/Decryption in the New Scheme

Let us define:

V_{trc} - A plaintext value located in table t, row r and column c.

$\mu : (N \times N \times N) \rightarrow N$ - a function that generates a number based on the database coordinates.

Enc_k - A function which encrypts a plaintext value with its coordinates.

$$Enc_k(V_{trc}) = E_k(V_{trc} \oplus \mu(t,r,c)) \quad \textbf{(2)}$$

Where k is the encryption key and E_k is a symmetric encryption function (e.g. DES, AES).

$\mathrm{X}_{\mathrm{trc}}$ - A ciphertext value located in table t, row r and column c.

$$\mathrm{X}_{\mathrm{trc}} = Enc_k(V_{trc}) \quad \textbf{(3)}$$

Dec_k - A function which decrypts a ciphertext value ($\mathrm{X}_{\mathrm{trc}}$) and discards its coordinates.

$$Dec_k(X_{trc}) = D_K(X_{trc}) \oplus \mu(T,R,C) = V_{trc} \quad \textbf{(4)}$$

Where k is the decryption key and D_k is a symmetric decryption function.

3.2 Data Integrity

Encryption ensures that a user not possessing the encryption key cannot modify a ciphertext value and predict the change in the plaintext value. Usually the range of valid plaintext values is significantly smaller than the whole range of possible plaintext values. Thus, the probability that an unauthorized change to a ciphertext value would result in a valid plaintext value is negligible. Therefore, unauthorized changes to ciphertext values are likely to be noticed at decryption time.

Substitution attacks as opposed to patterns matching attacks can not be prevented simply by using encryption. In the new scheme, each value is encrypted with its unique cell coordinates. Therefore, trying to decrypt a value with different cell coordinates (e.g. as a result of a substitution attack) would probably result in an invalid plaintext value.

If the range of valid plaintext values is not significantly smaller than the whole possible range, or invalid plaintext values cannot be distinguished from valid plaintext values, encryption has to be carried out as follows:

$$Enc_K(V_{trc}) = E_k(V_{trc} \parallel \mu(t,r,c)) \quad \textbf{(5)}$$

Since $\mu(t,r,c)$ is concatenated to the plaintext value before encryption, attempting to change the ciphertext value or trying to switch two ciphertext values would result in a corrupted $\mu(t,r,c)$[1] after decryption. Obviously, concatenating $\mu(t,r,c)$ results in data expansion.

3.3 Scheme Analysis

The new database encryption scheme satisfies the requirements mentioned in section 2:

1) The scheme security relies on the security of the encryption algorithm used. In order to reveal some database value it has to be decrypted using the correct key.
2) Encryption and decryption are fast operations and are mandatory in any database encryption scheme. The proposed implementation adds the overhead of a Xor operation and μ computation which are negligible compared to encryption.
3) Using encryption algorithms such as DES or AES which are based on encrypting blocks of data results in value expansion (in many cases this expansion is negligible).
4) The basic element of reference is a database cell. Operations on a cell do not depend on or have any effect on other cells.
5) The proposed scheme facilitates subschema implementation. Since each cell is encrypted separately, each column can be encrypted under a different key[2].
6) The new scheme prevents patterns matching attacks since there is no correlation between a plaintext value and a ciphertext value (achieved by using encryption) and there is no correlation between ciphertext values (achieved by using μ before encryption). Substitution attacks are also prevented as discussed in section 3.2.
7) Unauthorized manipulation on the encrypted data without the encryption key would be noticed at decryption time. (see section 3.2)
8) As the basic element of reference is a database cell, it is possible to recover information from partially completed records (records with null values) in the same way as it is recovered from full records.
9) The new scheme complies with the structure preserving requirements as the basic element of reference is a database cell.

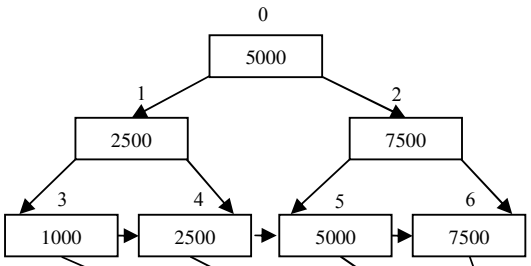
4 The Desired Properties of a Secure Indexing Scheme

An index is a data structure supporting efficient access to data and indexes are frequently used in databases. Most commercial databases even create a default index on the primary-key columns. Most databases implement indexes using a B+-Tree which

[1] μ implementation is discussed in section 6.2.
[2] Key management is discussed in section 6.3.

is a data structure maintaining an ordered set of values and supporting efficient operations on this set such as search, insert, update and delete.

a) An Index Constructed on Column C in Table T



b) Table Representation of the Index

ID	Struct.	Data
0	1,2	5000
1	3,4	2500
2	5,6	7500
3	4	1000, 2
4	5	2500, 1
5	6	5000, 0
6	-	7500, 3

c) Table T

Row	C
0	5000
1	2500
2	1000
3	7500

Fig. 2. An example of a database index.

Figure 2 illustrates a database index which is constructed on column C in table T and is implemented as a B+-Tree. A graphical representation of the B+-Tree is given in figure 2a; a table representation of the B+-Tree is given in figure 2b and table T is given in figure 2c. Figure 2b sharpens the separation between the index structure and its data.

A secure index in an encrypted database has to comply with the following requirements:

1) No information about the database plaintext values can be learned from the index.
2) The secure index should not reduce the efficiency of data access.
3) The secure index should not reduce the efficiency of insert, update and delete operations.
4) The secure index should not have a significantly greater volume than an ordinary index.
5) The secure index structure should not differ from a standard index. In this way, a DBA can manage the index without the encryption key.

A trivial approach which constructs an index over the plaintext values would reduce security since the plaintext values are exposed. Another approach would con-

struct the index over the database ciphertext values. In this approach, executing equality queries is possible but executing range queries is a problem. This approach would expose the index to patterns matching attacks since equal plaintext values are encrypted to equal ciphertext values. Moreover, since executing range queries is a problem, Oracle does not support encrypting indexed data [20].

In the next section, a new indexing scheme which overcomes the shortcomings of existing indexing schemes is presented.

5 A New Database Indexing Scheme

Several indexing schemes for encrypted databases were proposed [15, 18, 17, 21] that fulfill most of the requirements described in section 4 but none preserve the index structure. We claim that there should be a separation between data and structure. For example, A DBA should be able to manage database indexes without the need of decrypting its values.

We suggest a new database indexing scheme which preserves the index structure where each index value is the result of encrypting a plaintext value in the database concatenated with its row-id. This ensures that there is no correlation between the index values and the database ciphertext values[3]. Furthermore, the index does not reveal the statistics or order of the database values.

5.1 Index Construction in the New Scheme

In order to construct an index, a set of values and a function determining the order[4] of these values are needed.

Let us define:

C - An encrypted database column that was encrypted as defined in section 3.1.

C_p - The column obtained from decrypting column C :

$$Dec_k(x_{trc}) \in C_p \longleftrightarrow x_{trc} \in C \tag{6}$$

Where Dec_k is the decryption function defined in section 3.1.

C_i - The column obtained from encrypting values in C_p concatenated with their row-ids:

$$E_k(V_{trc} \parallel r) \in C_i \longleftrightarrow V_{trc} \in C_p \tag{7}$$

---

[3] If the database is encrypted as described in section 3.2, then μ should not be implemented as $\mu(t,r,c) = r$ since there will be a strong correlation between the index values and the database encrypted values.

[4] Some indexes require only an equality function and not an order function to be constructed. In this case, the term "order" in this section can be replaced by the term "equality".

Where k is the encryption key, E_k is an encryption function and r is the row id.

$\lambda_k : C_i \rightarrow C_p$ - A function which decrypts a value in C_i (using key k) and discards its row-id:

$$\lambda_k(x) = Discard(D_k(x), |r|) \quad \textbf{(8)}$$

Where k is the decryption key, D_k is a decryption function, r is the row-id, $|r|$ is the length of r in bits, and $Discard(v,n)$ stands for discarding the n rightmost bits of v.

R_p - The values in C_p are ordered by the relation R_p:

$$(x, y) \in R_p \longleftrightarrow x, y \in C_p And (x \leq y) \quad \textbf{(9)}$$

R_i - The values in C_i are ordered by the relation R_i:

$$(x, y) \in R_i \longleftrightarrow x, y \in C_i And (\lambda_k(x), \lambda_k(y)) \in R_p \quad \textbf{(10)}$$

The new index will be constructed based on the values in C_i, using the relation R_i as an order function.

a) Encryption of Table T in the New Scheme

Row	C
0	$E_k(5000 \oplus \mu(T,0,C))$
1	$E_k(2500 \oplus \mu(T,1,C))$
2	$E_k(1000 \oplus \mu(T,2,C))$
3	$E_k(7500 \oplus \mu(T,3,C))$

b) Encryption of the Index in the New Scheme

ID	Struct.	Data
0	1,2	E_k (5000 ‖ 0)
1	3,4	E_k (2500 ‖ 1)
2	5,6	E_k (7500 ‖ 3)
3	4	E_k (1000 ‖ 2)
4	5	E_k (2500 ‖ 1)
5	6	E_k (5000 ‖ 0)
6	-	E_k (7500 ‖ 3)

Fig. 3. Encryption in the new scheme.

Figure 3 illustrates encryption of the table and the index which were illustrated in figure 2 using the new schemes. Figure 3a describes the encryption of the table in the new scheme where each cell is encrypted with its coordinates. Figure 3b describes the encryption of the index where each index value is the result of encrypting a database plaintext value concatenated with its row-id. It is easy to see that the table and index structure are not changed by the encryption process.

5.2 Executing a Query in the New Scheme

The following SQL query illustrates the retrieval of all rows in table T, which their values in column C are greater or equal to V:

```
SELECT * FROM T WHERE T.C>=V
```
(11)

The following pseudo code illustrates the retrieval of row-ids of rows which answer the above query. The pseudo code assumes that the index is implemented as a binary B+-Tree.

```
INPUT: A table T, a column C and a value V.
OUTPUT: A collection of row-ids.

X := getIndex(T, C).getRootNode();

While X is not a leaf Do
   If X.getData().getValue()<V Then
      X := X.getRightSonNode();
   Else
      X := X.getLeftSonNode();
   End If;
End While;

RESULT := ϕ;

While X.getData().getValue()<V Do
   X := X.getRightSiblingNode();
End While;

While X is not null Do
   RESULT := RESULT∪{X.getData().getRowId()};
   X := X.getRightSiblingNode();
End While;

Return RESULT;
```

Each *node* in the index which is not a *leaf* has a *left son node*, a *right son node* and a *data* which stores a value. Each *leaf* in the index has a *right sibling node* and a *data* which stores a value and a row-id.

In the new scheme the data in each index node is an encryption of a database value concatenated with its row-id. Thus, the functions getValue() and getRowId() need to be given a new implementation in order to support the new indexing scheme. However, the above pseudo code stands without any change.

5.3 Index Integrity

In the new scheme, a substitution attack which attempts to substitute index values can be carried out without being noticed at decryption time. If it is possible to maintain a unique position for each value in the index, this kind of attack can be eliminated using

a technique similar to the one proposed in section 3 where each value is encrypted with its unique position.

a) Maintaining Data Integrity of Table T

Row	C
0	$E_k(5000 \| \mu(T,0,C))$
1	$E_k(2500 \| \mu(T,1,C))$
2	$E_k(1000 \| \mu(T,2,C))$
3	$E_k(7500 \| \mu(T,3,C))$

b) Maintaining Data Integrity of the Index

ID	Struct.	Data
0	1,2	$E_k(5000 \| 0)$
1	3,4	$E_k(2500 \| 1)$
2	5,6	$E_k(7500 \| 2)$
3	4	$E_k((1000,2) \| 3)$
4	5	$E_k((2500,1) \| 4)$
5	6	$E_k((5000,0) \| 5)$
6	-	$E_k((7500,3) \| 6)$

Fig. 4. Maintaining data integrity.

Figure 4 illustrates data integrity maintenance of the table and the index which were illustrated in figure 2. Figure 4a describes data integrity maintenance of the table as suggested in section 3.2. Figure 4b describes data integrity maintenance of the index where each index value is concatenated to its unique position in the index (ID) and then encrypted.

We argue that without changing the index structure and affecting its efficiency, maintaining a unique position for each value in the index is not a trivial matter.

5.4 Scheme Analysis

The new index implementation on an ordered set of values is identical to the ordinary index implementation. The only differences between the ordinary index and the new one are the set of values and the order function defined on them.

The new index complies with the requirements mentioned in section 4:

1) Since the values in the index are encrypted and unique (achieved by concatenating row-id) there is no correlation between them as to the column ciphertext values, or the column plaintext values. Therefore, no information is revealed on the database data by the new index.
2) The order function is implemented in a time complexity of $O(1)$ since decryption and discarding bits are implemented in a time complexity of $O(1)$. Therefore, data access using the proposed index is as efficient as with an ordinary index.
3) Determining the order of two values is implemented in a time complexity of $O(1)$. Therefore, the delete operation is as efficient as in an ordinary index. En-

crypting a new value is implemented in a time complexity of $O(1)$, thus the efficiency of insert and update operations is not changed.

4) Each value in the new index is a result of encrypting a database plaintext value concatenated with its row-id, therefore the space added for each node in the new index is fixed. Thus, the index space complexity remains the same.
5) The new index structure remains the same and only its data is modified. Thus, any administrative work on the index can be carried out without the need of decrypting the index values.

6 Performance and Implementation Issues

Implementing the new schemes requires careful consideration. Several performance and implementation issues are discussed in this section.

6.1 Stable Cell Coordinates

The proposed scheme assumes that cell coordinates are stable. That is, insert, update and delete operations do not change the coordinates of existing cells. However, if a database reorganization process changes cell coordinates, all affected cells are to be re-encrypted with their new coordinates and the index updated respectively.

A naïve implementation which uses the row number in the table as the row-id, proves to be limited in this respect as row numbers are affected by insert and delete operations. In the Oracle database, for example, cell coordinates are stable.

6.2 Implementing a Secure μ Function

As defined in section 3.2, the values in the database are encrypted as follows:

$$Enc_K(V_{trc}) = E_k(V_{trc} \parallel \mu(t,r,c)) \tag{12}$$

A secure implementation of μ would generate different numbers for different coordinates:

$$(t_1,r_1,c_1) \neq (t_2,r_2,c_2) \longleftrightarrow \mu(t_1,r_1,c_1) \neq \mu(t_2,r_2,c_2) \tag{13}$$

Unfortunately, generating a unique number for each database coordinates may result in considerable data expansion. An alternative implementation reducing the data expansion may result in collisions. Assume that there are two cells, which μ generates two equal values for their coordinates:

$$\begin{aligned} &\exists t_1,r_1,c_1,t_2,r_2,c_2 \mid \\ &[(t_1,r_1,c_1) \neq (t_2,r_2,c_2)] \wedge [\mu(t_1,r_1,c_1) = \mu(t_2,r_2,c_2)] \end{aligned} \tag{14}$$

It is possible to substitute the ciphertext values of these cells ($x_{t_1r_1c_1}$ and $x_{t_2r_2c_2}$) without μ being corrupted at decryption time. If it is difficult to find two cells such as those mentioned above, this kind of attack can be prevented. This can be achieved by using a collision free hash function.

6.3 Key Management

Databases contain information of different sensitivity degrees that have to be selectively shared between a large numbers of users. The proposed scheme facilitates subschema implementation since each column can be encrypted with a different key. Encrypting each column with a different key, results in a large number of keys for each legitimate user. However, using the approach proposed in [22] can reduce the number of keys. It is suggested in [22] how the smallest elements which can be encrypted using the same key according to the access control policy can be found. Thus, the keys are generated according to the access control policy in order to keep their number minimal. This approach can be incorporated in the proposed scheme in order to encrypt sets of columns with the same key in accordance with the database access control policy.

6.4 Performance

In the new scheme, all conventional algorithms remain the same since the structure of the database remains the same. This ensures that the only overhead of the new scheme is that of encryption and decryption operations.

7. Conclusions

In this paper, a new structure preserving scheme for database encryption has been presented. In the new scheme, each database cell is encrypted with its unique position and this guarantees that patterns matching and substitution attacks cannot succeed, thus, guaranteeing information confidentiality and data integrity.

A new database indexing scheme that does not reveal any information on the database plaintext values was proposed. In the new scheme index values are encrypted with a unique number (the row-id of the database value) in order to eliminate patterns matching attacks and any correlation between index and database values. Ensuring index integrity is possible if an index position can be attached to each index value by simply using a technique similar to the one used for table encryption.

The new schemes do not impose any changes on the database structure, thus enabling a DBA to manage the encrypted database as any other non-encrypted database. Furthermore, implementing the new scheme in existing applications does not entail modifying the queries.

References

1. Date, C.J.: An Introduction to Database Systems. Vol. 1, Fifth Edition. Addison Wesley, Massachusetts (1990)
2. Fernandez , E.B., Summers, R.C. and Wood C.: Database Security and Integrity. Addison-Wesley, Massachusetts, (1980)
3. Coper, J.A.: Computer & Communication Security: Strategies for the 1990s. McGraw-Hill, New York (1989)
4. Conway, R.W., Maxwell, W.L. and Morgan, H.L.: On the implementation of security measures in information systems. Communications of the ACM 15(4) (1972) 211-220
5. Graham, G.S. and Denning, P.J.: Protection - Principles and practice. Proc. Spring Jt. Computer Conf., Vol. 40, AFIPS 417-429, Montrale, N.J. (1972)
6. Hwang, M.S. and Yang, W.P.: A new dynamic access control scheme based on subject-object-list. Data and Knowledge Engineering 14(1) (1994) 45-56
7. Garvey, C. and Wu, A.: ASD-Views. Proc. IEEE Symposium on Security and Privacy, Oakland, California (1988) 85-95
8. Lunt, T.F., Denning, D.E., Schell, R.R., Heckman, M. and Shockley, W.R.: The SeaView security model. IEEE Trans. on Software Engineering, SE-16(6) (1990) 593-607
9. Stachour, P.D. and Thuraisingham, B.: Design of LDV: A multilevel secure relational database management system, IEEE Trans. on Knowledge and Data Engineering 2(2) (1990) 190-209
10. National Bureau of Standards. Data Encryption Standard. FIPS, NBS (1977)
11. Rivest, R.L., Shamir, A. and Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM 21(2) (1978) 120-126
12. Smid, M.E. and Branstad, D.K.: The data encryption standard: past and future. Proc. IEEE 76(5) (1988) 550-559
13. Davida, G.I., Wells, D.L., and Kam, J.B.: A Database Encryption System with Subkeys. ACM Trans. Database Syst. 6 (1981) 312-328
14. Min-Shiang, H. and Wei-Pang, Y.: Multilevel secure database encryption with subkeys. Data and Knowledge Engineering 22 (1997) 117-131
15. Damiani, E., De Capitani diVimercati, S., Jajodia, S., Paraboschi, S. and Samarati, P.: Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs. CCS'03, Washington (2003) 27–31
16. Denning, D.E.: Cryptography and Data Security. Addison-Wesley, Massachusetts (1982)
17. Hacigümüs, H., Iyer, B., Li, C., and Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In Proc. of the ACM SIGMOD'2002, Madison, Wisconsin, USA (2002)
18. Iyer, B., Mehrotra, S., Mykletun, E., Tsudik, G. and Wu, Y.: A Framework for Efficient Storage Security in RDBMS. E. Bertino et al. (Eds.): EDBT 2004, LNCS 2992 (2004) 147–164
19. Bouganim, L. and Pucheral, P.: Chip-secured data access: Confidential data on untrusted servers. In Proc. of the 28th International Conference on Very Large Data Bases, Hong Kong, China (2002) 131–142
20. Database Encryption in Oracle9i™. An Oracle Technical White Paper (2001)
21. Bayer, R. and Metzger, J.K.: On the Encipherment of Search Trees and Random Access Files. ACM Trans Database Systems, Vol. 1 (1976) 37-52
22. Bertino, E. and Ferrari, E.: Secure and Selective Dissemination of XML Documents. ACM Transactions on Information and System Security Vol. 5 No. 3 (2002) 290–331
23. Hwang, M.S. and Yang, W.P.: A two-phase encryption scheme for enhancing database security. J. Systems and Software 31(12) (1995) 257-265